IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 3, pp. 270–293. DOI:10.46586/tches.v2023.i3.270-293

Efficient Regression-Based Linear Discriminant Analysis for Side-Channel Security Evaluations

Towards Analytical Attacks against 32-bit Implementations

Gaëtan Cassiers[1*], Henri Devillez[2],
François-Xavier Standaert[2] and Balazs Udvarhelyi[2]

[1] TU Graz, Graz, Austria, firstname.lastname@iaik.tugraz.at
[2] UCLouvain, ICTEAM, Crypto Group, Louvain-la-Neuve, Belgium
firstname.lastname@uclouvain.be

Abstract. 32-bit software implementations become increasingly popular for embedded security applications. As a result, profiling 32-bit target intermediate values becomes increasingly needed to evaluate their side-channel security. This implies the need of statistical tools that can deal with long traces and large number of classes. While there are good options to solve these issues separately (e.g., linear regression and linear discriminant analysis), the current state of the art lacks efficient tools to solve them jointly. To the best of our knowledge, the best-known option is to fragment the profiling in smaller parts, which is suboptimal from the information theoretic viewpoint. In this paper, we therefore revisit regression-based linear discriminant analysis, which combines linear regression and linear discriminant analysis, and improve its efficiency so that it can be used for profiling long traces corresponding to 32-bit implementations. Besides introducing the optimizations needed for this purpose, we show how to use regression-based linear discriminant analysis in order to obtain efficient bounds for the perceived information, an information theoretic metric characterizing the security of an implementation against profiled attacks. We also combine this tool with optimizations of soft analytical side-channel attack that apply to bitslice implementations. We use these results to attack a 32-bit implementation of ISAP instantiated with Ascon's permutation, and show that breaking the initialization of its re-keying in one trace is feasible for determined adversaries.

Keywords: Linear Regression · Linear Discriminant Analysis · Belief Propagation

1 Introduction

Single-trace side-channel attacks have recently gained interest due to their increasing relevance for different emerging applications in embedded secure design. For example, Kannwischer et al. showed their impact for post-quantum cryptographic algorithms [KPP20] and Bellizia et al. highlighted their importance for leakage-resistant modes of operation [BBC+20]: on the one hand, they are the main attack vector to break the confidentiality of schemes leveraging an internal re-keying, like Ascon [DEMS21], Spook [BBB+20] or TEDT [BGP+20], assuming that the initialization and finalization are strongly protected (e.g., with masking); on the other hand, schemes like ISAP even reduce their long term security (and integrity) to the assumption that such attacks are hard [DEM+20].

Looking at these references, it appears that single-trace attacks are in general easy against 8-bit implementations: Soft Analytical Side-Channel Attacks (SASCA) are indeed

* The work was done in parts while being a Research Fellow of the Belgian Fund for Scientific Research (FNRS-F.R.S.) with UCLouvain and in parts while being with Lamarr Security Research.

Received: 2023-01-15 Accepted: 2023-03-15 Published: 2023-06-09

readily applicable in this context [VGS14]. By contrast, they are considered hard in the case of large parallel (e.g., 128-bit hardware) implementations, due to the limited side-channel signal that then can be collected [USS+20, BMPS21]. In-between these two extremes, the situation is currently more fuzzy. For example, Kannwischer et al. only show simulated attacks against 32-bit implementations under a Hamming weight assumption (which reduces the cardinality of the leakages to consider from 2^{32} to 33), and Bellizia et al. could only target a 16-bit implementation of Keccak. This leads to an uncomfortable situation given the importance of 32-bit embedded microcontrollers in the benchmarking of side-channel resistant lightweight and post-quantum cryptography.[1] To the best of our knowledge, the only work which succeeded targeting a concrete 32-bit implementation so far is the recent Fragment Template (FT) attack of You and Kuhn [YK21].

Intuitively, the main reason that makes such attacks challenging is their profiling complexity, which is due to the combined difficulty of dealing with large intermediate values and long leakage traces that makes the direct application of Chari et al.'s template attacks hardly realistic [CRR02]. More precisely, when taken separately, these two issues have good solutions. Profiling large target intermediate values is known to be efficiently performed thanks to Linear Regression (LR), as introduced by Schindler et al. [SLP05]. And the problem of estimating the covariance of long leakage traces can be mitigated thanks to dimensionality reduction, e.g., with Linear Discriminant Analysis (LDA) [SA08]. The only attempt to combine these two techniques observed that LR and LDA form an effective combination (significantly better than, e.g., LR and principal component analysis), but concluded that estimating 32-bit templates was not practical [CK14], which actually led (a subset of) the authors to investigate FT attacks. The latter rather considers the independent modeling of (e.g., 8-bit) fragments of target intermediate variables. But this approach is conceptually unsatisfying, since it implies that a part of the signal is considered as algorithmic noise, which is suboptimal from the information extraction viewpoint.

Based on this state of the art, the main goal of this work is to optimize Regression-based Linear Discriminant Analysis, next coined RLDA, so that 32-bit target intermediate values can be efficiently profiled. Our contributions in this respect are threefold. First, we detail the optimized technique and how to implement it efficiently. Second, we show how to use RLDA in order to rapidly bound the Perceived Information (PI), an information theoretic metric that characterizes the (data) complexity of a profiled side-channel attack [DFS19, dCGRP19, BHM+19]. Third, we describe how to combine RLDA with SASCA and optimize SASCA in order to make it more efficiently applicable against the 32-bit target intermediate values of a bitslice cryptographic implementation.

As a result, we exhibit a concrete attack that breaks the initialization of the re-keying used by ISAP implemented in a 32-bit bitslice fashion. We additionally show that RLDA consistently outperforms FT. Besides applying such powerful profiling methods for the first time, our results highight new research challenges towards their further optimization, in particular for the (heuristic) SASCA. Given the potential (algorithmic and technological) improvements of RLDA in the future, these results also suggest that the security of leakage-resistant pseudorandom functions (like used in ISAP) based on 32-bit unprotected implementations may not be sufficient against determined adversaries.

We note that RLDA can naturally be applied in the context of multi-trace attacks (a.k.a. differential side-channel attacks) as well. Due to its efficiency and wide applicability, we hope it can become a workhorse for profiled side-channel security evaluations.

Related works. As an alternative to "classical" profiled attacks based on templates, linear regression and dimensionality reduction (see references in the above state of the art), a large amount of recent works investigate the applicability of machine learning and

[1] https://csrc.nist.gov/Projects/lightweight-cryptography, https://csrc.nist.gov/projects/post-quantum-cryptography.

deep learning to the context of profiled side-channel analysis. We cite [HZ12, MPP16] for early results in this direction and [CDP17, MDP20] for more recent ones. Like classical template attacks, these works are hardly applicable to large target intermediate values, due to profiling complexity reasons. One notable exception is the neural estimation of the mutual information proposed in [CLM20]. However, contrary to our work, the latter can only be used to estimate the security level of an implementation, not to mount an attack (in that sense it is only similar to our PI bound of Subsection 3.2). We also note the recent [KDB+22] which performs analytical attacks against implementations of stream cipher based on a Hamming weight assumption (to reduce profiling complexity).

2 Background

In this section, we first remind the LR, LDA and (baseline) RLDA techniques for profiled attacks, together with the FT which has been proposed as another way to solve the problem we tackle.[2] We then introduce the PI metric that we will use to quantify the quality of a profiled model. We finally describe the SASCA that we will use as a mean to exploit the leakage of many target intermediate values in a leaking implementation.

Notations. We use capital letters X for random variables, bold letters $\boldsymbol{x}$ for vectors, capital bold letters $\boldsymbol{X}$ for matrices and calligraphic letters $\mathcal{X}$ for sets. We additionally use the notation $\boldsymbol{X} = (\boldsymbol{x}_1 \dots \boldsymbol{x}_n)$ to denote the matrix whose columns are the vectors $\boldsymbol{x}_i$, and the notation $\mathbb{1}_n$ to denote the vector of length n whose coordinates are all 1.

We use the following conventions: n_s is the number of samples in a trace, n_p and n_a are respectively the number of profiling and attack traces, b is the number of bits of the profiled variable, and p is the LDA reduced subspace dimensionality.

We denote by $\hat{\mathsf{f}}^{\mathsf{M}}[\boldsymbol{l}^*|X = x]$ the estimation by model M (e.g., LR, LDA, ...) of the probability density of the leakage trace $\boldsymbol{l}^*$ conditioned on the intermediate variable X having value x. Then, the distribution of X is always computed using the Bayes rule (assuming that the prior distribution if X is known):

$$\hat{\mathsf{f}}^{\mathsf{M}}[X = x|\boldsymbol{l}^*] = \frac{\hat{\mathsf{f}}^{\mathsf{M}}[\boldsymbol{l}^*|X = x] \Pr[X = x]}{\sum_{x' \in \mathcal{X}} \hat{\mathsf{f}}^{\mathsf{M}}[\boldsymbol{l}^*|X = x^*] \Pr[X = x']}. \quad (1)$$

2.1 Regression-based leakage profiling

LR-based profiled attacks have been introduced by Schindler et al. [SLP05]. They can be viewed as a variant of Chari et al.'s template attacks [CRR02], where the deterministic part of the leakage function is expressed as a linear combination of basis functions, in order to reduce the number of profiling traces. For every time sample s in a leakage trace, instead of estimating this deterministic part thanks to the mean of the sample for every class (i.e., every possible value $x \in \mathcal{X} = \mathbb{Z}_{2^b}$ of the target variable), the LR-based attack fits a weighted sum of n_b basis functions denoted as β_i:

$$\mathsf{m}_s(x) = \boldsymbol{a}_s^T \cdot \boldsymbol{\beta}(x) = \sum_{i=0}^{n_b-1} a_{i,s} \beta_i(x),$$

where the $a_{i,s}$'s are the weights of the model learned by profiling. A simple choice for the basis functions is the linear basis, where each function β_i corresponds to a single bit of the

[2] The authors of [SLP05] and [CK14] respectively denote LR attacks and RLDA as stochastic attacks and SLDA, where the S stands for stochastic. We use the term regression-based and therefore the letter R in our acronyms, which is more descriptive and in line with statistical textbooks terminology.

target value x:

$$\beta_i(x) = \begin{cases} 1 & \text{if } i = 0, \\ 1 & \text{if } \lfloor x/2^{i-1} \rfloor \mod 2 = 1, \\ -1 & \text{otherwise,} \end{cases} \tag{2}$$

with $n_b = b + 1$. The profiling step then collects a set of n_p traces $\boldsymbol{L} = (\boldsymbol{l}_1 \cdots \boldsymbol{l}_{n_p})$ and their corresponding values $\boldsymbol{x} = (x^1, x^2, \ldots, x^{n_p}) \in \mathcal{X}^{n_p}$. Letting $\boldsymbol{l}'_s$ be the sth row of $\boldsymbol{L}$ and $\boldsymbol{\beta}(\boldsymbol{x}) = \big(\boldsymbol{\beta}(x^1) \ldots \boldsymbol{\beta}(x^{n_p})\big)$, it minimizes the norm of the residual:

$$\boldsymbol{r}_s^T = (\boldsymbol{l}'_s)^T - \boldsymbol{a}_s^T \cdot \boldsymbol{\beta}(\boldsymbol{x}). \tag{3}$$

The minimization of $\|\boldsymbol{r}_s\|$ is an ordinary linear least squares problem that can be transformed into a small linear system using the normal equations:

$$(\boldsymbol{l}'_s)^T \cdot \boldsymbol{\beta}(\boldsymbol{x})^T = \boldsymbol{a}_s^T \cdot \boldsymbol{\beta}(\boldsymbol{x}) \cdot \boldsymbol{\beta}(\boldsymbol{x})^T, \tag{4}$$

for each time sample $s \in \{1, \ldots, n_s\}$. Once $\mathsf{m}_s(x)$ is computed, it is used as the mean for a pooled template [CK13], i.e., we estimate a single approximate covariance matrix for all the traces $\hat{\Sigma}$, computed as the empirical covariance of the residual $\boldsymbol{R} = (\boldsymbol{r}_1 \cdots \boldsymbol{r}_{n_s})^T$:

$$\hat{\Sigma} = \frac{1}{n_p} \boldsymbol{R}\boldsymbol{R}^T. \tag{5}$$

(There is no need to subtract the mean since $\boldsymbol{r}$ has mean 0 by construction.)

Finally, the modeled probability density for a new leakage trace $\boldsymbol{l}^*$ is:

$$\hat{\mathsf{f}}^{\mathsf{LR}}[\boldsymbol{l}^* | X = x] = \frac{1}{\sqrt{(2\pi)^n \left|\hat{\Sigma}\right|}} \exp\left(-\frac{1}{2}(\boldsymbol{l}^* - \mathbf{m}(x))^T \hat{\Sigma}^{-1} (\boldsymbol{l}^* - \mathbf{m}(x))\right), \tag{6}$$

where $\mathbf{m}(x) = (\mathsf{m}_1(x), \ldots, \mathsf{m}_{n_s}(x))$. The conditional distribution $\hat{\mathsf{f}}^{\mathsf{LR}}[X | \boldsymbol{l}^*]$ can in turn be derived from the probability density function $\hat{\mathsf{p}}^{\mathsf{LR}}[\boldsymbol{l}^* | X]$ using Bayes' rule.

2.2 Linear discriminant analysis

The exploitation of long traces using (pooled) Gaussian templates or LR-based attacks can be resource-intensive (i.e., require many profiling traces and a long computation time) due to the need to accurately estimate and invert a large covariance matrix $\hat{\Sigma}$. A good solution to mitigate this problem is to apply Fisher's LDA in order to reduce the dimensionality of the trace while preserving (most of its) useful content [SA08]. In a nutshell, LDA finds a linear projection to a fixed-dimensionality subspace that maximizes the side-channel Signal-to-Noise Ratio (SNR) [Man04] in that subspace. Formally, it finds the projection matrix $\boldsymbol{W} \in \mathbb{R}^{n_s \times p}$ that maximizes the ratio between the inter-class scatter $\boldsymbol{S_B}$ and the intra-class scatter $\boldsymbol{S_W}$, that is [DHS01]:

$$\arg\max_{\boldsymbol{W}} \frac{\left|\boldsymbol{W}^T \boldsymbol{S_B} \boldsymbol{W}\right|}{\left|\boldsymbol{W}^T \boldsymbol{S_W} \boldsymbol{W}\right|}.$$

Given a set of traces $\mathcal{L}$, let $\mathcal{L}(x)$ be the subset of traces in $\mathcal{L}$ for which $X = x$. The scatter matrices can then be computed as:

$$\begin{aligned}
\hat{\boldsymbol{\mu}}(x) &= \frac{1}{|\mathcal{L}(x)|} \sum_{\boldsymbol{l} \in \mathcal{L}(x)} \boldsymbol{l}, \\
\hat{\boldsymbol{\mu}} &= \frac{1}{|\mathcal{L}|} \sum_{\boldsymbol{l} \in \mathcal{L}} \boldsymbol{l}, \\
\boldsymbol{S_B} &= \sum_{x \in \mathcal{X}} |\mathcal{L}(x)| \, (\hat{\boldsymbol{\mu}}(x) - \hat{\boldsymbol{\mu}})(\hat{\boldsymbol{\mu}}(x) - \hat{\boldsymbol{\mu}})^T, \\
\boldsymbol{S_W} &= \sum_{x \in \mathcal{X}} \sum_{\boldsymbol{l} \in \mathcal{L}(x)} (\boldsymbol{l} - \hat{\boldsymbol{\mu}}(x))(\boldsymbol{l} - \hat{\boldsymbol{\mu}}(x))^T,
\end{aligned}$$

and the maximization can be rewritten as the generalized eigendecomposition problem:

$$\boldsymbol{S_B} \boldsymbol{w}_i = \lambda_i \boldsymbol{S_W} \boldsymbol{w}_i,$$

where $\boldsymbol{w}_i$ is the ith column of $\boldsymbol{W}$ (we consider only the p largest eigenvalues and the associated eigenvectors). Once the projection is known, the traces are projected in the subspace and any model can be built from the projected traces. In the particular case of pooled Gaussian templates, the model becomes:

$$\hat{\mathsf{f}}^{\mathsf{LDA}}[\boldsymbol{l}^* | X = x] = \frac{1}{\sqrt{(2\pi)^n \left| \hat{\Sigma}_{\boldsymbol{W}} \right|}} \exp\left(-\frac{1}{2} (\boldsymbol{W}^T \boldsymbol{l}^* - \boldsymbol{W}^T \hat{\boldsymbol{\mu}}(x))^T \hat{\Sigma}_{\boldsymbol{W}}^{-1} (\boldsymbol{W}^T \boldsymbol{l}^* - \boldsymbol{W}^T \hat{\boldsymbol{\mu}}(x)) \right),$$

with $\hat{\Sigma}_{\boldsymbol{W}} = |\mathcal{L}|^{-1} \boldsymbol{W}^T \boldsymbol{S_W} \boldsymbol{W}$. If the number of dimensions after projection p is small (i.e., $p \ll n_s$, with n_s the number of samples in the raw traces), the LDA uses only a small part of the spectrum of the trace's covariance (or equivalently scatter) matrix, which converges fairly fast, hence mitigates the "big covariance estimation" problem.

2.3 Baseline regression-based linear discriminant analysis

The previous sections outline the respective merits of LR-based attacks and LDA: the efficient profiling of large states (i.e., many bits) for LR-based attacks and the efficient handling of long traces (i.e., many samples) for LDA. Since our goal is to efficiently profile long traces with large states, combining both methods is a natural solution.

In a nutshell, the baseline RLDA model proposed in [CK14] first computes the coefficients $\boldsymbol{a}_s$ of the linear regression for each time sample s in the trace, following the technique presented in Subsection 2.1. Then, it performs the LDA, but uses the regressed class means $\mathbf{m}(x)$ instead of the directly estimated means $\hat{\boldsymbol{\mu}}(x)$, both in the computation of the scatter matrices $\boldsymbol{S_W}$ and $\boldsymbol{S_B}$, and in the final Gaussian probability formula.

Formally, after computing the regression as in Subsection 2.1, we define the coefficient matrix $\boldsymbol{A} = (\boldsymbol{a_0} \cdots \boldsymbol{a_{n_s}})^T$, such that $\mathbf{m}(x) = \boldsymbol{A}\boldsymbol{\beta}(x)$. Then, we define the regressed inter- and intra-class scatter matrices $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$ and $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ as:

$$\begin{aligned}
\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} &= \sum_{x \in \mathcal{X}} |\mathcal{L}(x)| \, (\mathbf{m}(x) - \hat{\boldsymbol{\mu}})(\mathbf{m}(x) - \hat{\boldsymbol{\mu}})^T, \\
\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} &= \sum_{x \in \mathcal{X}} \sum_{\boldsymbol{l} \in \mathcal{L}(x)} (\boldsymbol{l} - \mathbf{m}(x))(\boldsymbol{l} - \mathbf{m}(x))^T.
\end{aligned}$$

Let us remark that $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ can equivalently be defined as the scatter of the residual (which has mean 0): $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} = \boldsymbol{r}^T \boldsymbol{r}$ (see Equation 5). The following optimization problem is then

solved to obtain the projection matrix $\boldsymbol{W}^{\mathsf{reg}}$ for a chosen dimension parameter p:

$$\boldsymbol{W}^{\mathsf{reg}} = \underset{\boldsymbol{W} \in \mathbb{R}^{n_s \times p}}{\arg\max} \frac{|\boldsymbol{W}^T \boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} \boldsymbol{W}|}{|\boldsymbol{W}^T \boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} \boldsymbol{W}|}.$$

Finally, the pooled noise covariance matrix can be estimated from the intra-class scatter $\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}} = |\mathcal{L}|^{-1} \boldsymbol{W}^{\mathsf{reg}T} \boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} \boldsymbol{W}^{\mathsf{reg}}$, and the model becomes:

$$\hat{\mathsf{f}}^{\mathsf{RLDA}}[\boldsymbol{l}^* | X = x] = \alpha \exp\left(-\frac{1}{2}(\boldsymbol{W}^{\mathsf{reg}T}(\boldsymbol{l}^* - \mathbf{m}(x)))^T \hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}}^{-1} (\boldsymbol{W}^{\mathsf{reg}T}(\boldsymbol{l}^* - \mathbf{m}(x)))\right), \quad (7)$$

with $\alpha = 1/\sqrt{(2\pi)^p |\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}}|}$.

2.4 Fragment template attacks

As a solution to the challenges in running a RLDA on a 32-bit state [CK14], You and Kuhn introduced FT attacks [YK21]. Their main idea is to split the variable of interest into 2 (or more) smaller fragments which can be profiled thanks to RLDA and treat the rest of the state as (algorithmic) noise. In the attack phase, the information extracted from each fragment is then recombined. Concretely, they analyzed the leakage of 32-bit words thanks to four 8-bit fragments, and exploited this model in a SASCA against Keccak. Their SASCA is based on single-bit variables, therefore they additionally marginalize the 8-bit probabilities from each fragment to single-bit probabilities.[3]

2.5 Perceived information

The PI of a leakage model is a side-channel metric that characterizes the amount of information that can be extracted about a target variable X from leakage traces using a given model [RSV+11]. It is related to the success rate of an adversary that uses this model [DFS19, dCGRP19, MDP20], and it is a lower bound for the Mutual Information (MI) between the leakage and the target variable [BHM+19, MCHS22], which characterizes the worst-case security of an implementation and therefore makes an interesting tool for side-channel security evaluations [SMY09]. Concretely, and given a model $\hat{\mathsf{f}}[\boldsymbol{l}|x]$ and a set of leakage traces $\mathcal{L}'$, the PI can be estimated as:

$$\hat{\mathrm{PI}}(X, \boldsymbol{L}) = \mathrm{H}(X) + \frac{1}{|\mathcal{L}'|} \sum_{x \in \mathcal{X}} \sum_{\boldsymbol{l} \in \mathcal{L}'_x} \log_2 \hat{\mathsf{p}}[x|\boldsymbol{l}], \quad (8)$$

where $\mathrm{H}(X)$ is the entropy of X. The set of traces $\mathcal{L}'$ used to estimate the PI must differ from the set of traces used to fit the model, otherwise the estimation is biased.

2.6 Soft analytical side-channel attacks

The previous models (LR, LDA, RLDA and FT) can be used in order to extract information on a target intermediate variable X. However, a cryptographic implementation performs computations on multiple values that depend on a single key, and an adversary can then build models for multiple variables $X_1, \ldots, X_{n_v}$. The question therefore arises of how to exploit the leakage of all these operations. In particular, while it is in principle possible to compute the joint probability of the variables in order to find the maximum likelihood key (which would be the optimal solution), it is often infeasible for computational reasons, as it requires enumerating a state whose size is comparable to the key size. SASCA were

[3] Theoretically, it could be possible to improve their attack with an 8-bit factor graph. But the bit-level representation of Keccak is naturally more suited to a binary description.

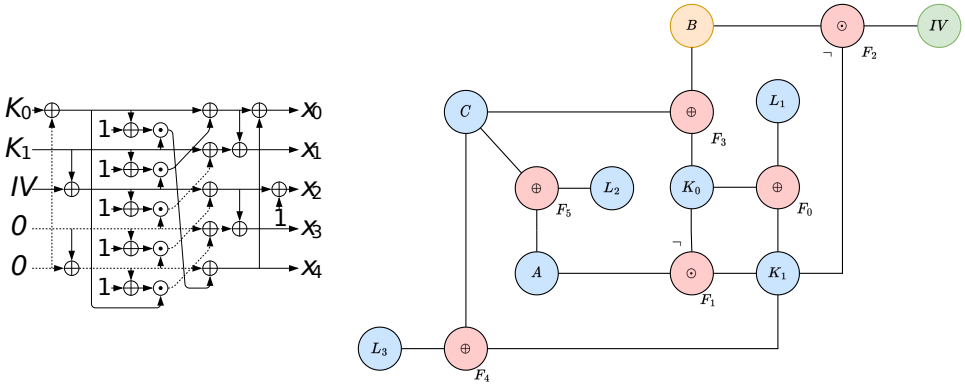

Figure 1: Ascon S-box for the initialization of ISAP' re-keying (the dashed lines are 0) and corresponding factor graph (the $\neg$, $\oplus$ and $\odot$ symbols are bitwise NOT, XOR and AND).

introduced to circumvent this problem [VGS14]. They leverage the known relationships between the variables manipulated by an implementation and approximate the joint distribution thanks to a message passing algorithm that exploits local inference rules.

In more details, a SASCA is based on a factor graph, that is a bipartite graph whose nodes are either variables X_i or relations R_i, and whose edges represent the involvement of a variable in a relation (e.g., a variable is the sum of two other variables). Each variable X_i has an *a priori* distribution, typically derived from the leakage, $\hat{\mathsf{p}}(X = x)$. For example, a factor graph for the Ascon S-box that we will use next is shown in Figure 1, where the blue nodes are leaky variables and the red nodes are relations. The yellow and green nodes are also variables but they are respectively non-profiled and known. The belief propagation algorithm works by computing probability distributions (over the domain of X) $P_X^{(t)}$, $m_{X\to R}^{(t)}$ and $m_{R\to X}^{(t)}$ for each variable X and for each edge connecting X to R. The algorithm is initialized by setting $m_{R\to X}^{(0)}$ to the uniform distribution for all edges. Next, the belief propagation performs a number of three-step iterations incrementing t that depend on the graph size and structure (starting at $t = 1$):

1. Intermediate state computation: for every variable node X,

$$P_X^{(t)}(x) = \hat{\mathsf{p}}(X = x) \prod_{R \in \partial X} m_{R\to X}^{(t-1)}(x),$$

   where ∂X denotes the set of neighbors of X.

2. From intermediate states to relations, the message is the probability of the state, except for the message received from the target relation:

$$m_{X\to R}^{(t)}(x) = P_X^{(t)} / m_{R\to X}^{(t-1)}(x).$$

3. From relations to intermediate states, the message is computed by summing the product of the incoming messages over all "compatible" values:

$$m_{R\to X}^{(t)}(x) = \sum_{x_i \in \mathcal{X}_i \text{ for } i=1,\ldots,k} \psi_R(x, x_1, \ldots, x_k) \prod_{i=1}^{k} m_{X\to R}^{(t)}(x_i),$$

   where $\{X_1, \ldots, X_k\} = \partial R \setminus X$ and $\psi_R(\cdot)$ is the relation that determines compatibility. The factor ψ_R equals 1 if the relation is satisfied, and 0 otherwise.

Running the belief propagation results in the final distributions $P_X^{(t^*)}$ for all variables of interest. Convergence in a bounded number of iterations is guaranteed if the factor graph is a tree. In the (frequent) case where it contains cycles, stopping the propagation after a few iteration is a common heuristic but does not guarantee convergence [VGS14].

3 Practical RLDA algorithms for large states

The previous section recalled how LR and LDA can be combined into a baseline RLDA, giving a conceptually powerful model that can be trained with relatively few traces. However, RLDA becomes computationally expensive when the number of bits in the state grows. Indeed, using the formulas of Subsection 2.3 has a complexity $\Theta\left(2^b n_s^2\right)$ for profiling, and $\Theta\left(2^b(b+p)n_s\right)$ for computing the distribution of a variable from one leakage trace.[4] In the following, we mitigate these performance bottlenecks by introducing improved solutions for efficient RLDA. We first describe a profiling algorithm with polynomial complexity, namely $\Theta\left(n_s^2(b+n_p)\right)$ computation and $\Theta\left(n_s^2\right)$ memory, with the additional advantage of being incremental: it can accumulate traces during their acquisition using a constant amount of memory. We next provide an optimized method for computing a probability distribution from the model in $\Theta\left((n_s+2^b)p\right)$ computations (i.e., we gain a factor n_s). In Section 3.2, we finally design a method allowing to quickly estimate the model quality with a new algorithm that computes the PI for large variables.

3.1 Efficient RLDA implementation

We next introduce our two steps to improve the efficiency of the RLDA. Precisely, we first highlight efficient and incremental formulas in order to compute the regression and the scatter matrices, and we discuss how to compute the projection. We next propose to slightly modify the projection in order to simplify the probability computations.

These efficient profiling computations are summarized in Algorithm 1.

Algorithm 1 Efficient RLDA profiling.

Input: Parameters n_s, b, n_p and p.
Input: Batches of profiling traces and associated intermediate values $(\boldsymbol{L}_1, \boldsymbol{x}_1), (\boldsymbol{L}_2, \boldsymbol{x}_2) \ldots$.
Output: $\boldsymbol{W}^{\mathsf{eff}}$ and $\boldsymbol{A}^{\mathsf{eff}}$ matrices for use in Equation 10.

$n_b \leftarrow b+1$
$\boldsymbol{B} \leftarrow \boldsymbol{0}^{n_b \times n_b}$; $\boldsymbol{C} \leftarrow \boldsymbol{0}^{n_s \times n_b}$; $\boldsymbol{S_L} \leftarrow \boldsymbol{0}^{n_s \times n_s}$
for each acquired batch $(\boldsymbol{L}_i, \boldsymbol{x}_i)$ **do**
    $\boldsymbol{B} \leftarrow \boldsymbol{B} + \boldsymbol{\beta}(\boldsymbol{x}_i)\boldsymbol{\beta}(\boldsymbol{x}_i)^T$
    $\boldsymbol{C} \leftarrow \boldsymbol{C} + \boldsymbol{L}_i\boldsymbol{\beta}(\boldsymbol{x}_i)^T$
    $\boldsymbol{S_L} \leftarrow \boldsymbol{S_L} + \boldsymbol{L}_i\boldsymbol{L}_i^T$
Solve the system $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$ for $\boldsymbol{A} \in \mathbb{R}^{n_s \times n_b}$.
$\hat{\boldsymbol{\mu}} \leftarrow \boldsymbol{C}_{*,1}/n_p$ ($\boldsymbol{C}_{*,1}$ is the first column of $\boldsymbol{C}$).
Compute $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$ and $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ using Proposition 1 and Proposition 2.
Find the p largest eigenvalues and associated eigenvectors $\boldsymbol{W}^{\mathsf{reg}}$ that solve Equation 9.
Compute the symmetric eigendecomposition $\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$ of $\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}} = \boldsymbol{W}^{\mathsf{reg}T}\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}\boldsymbol{W}^{\mathsf{reg}}/n_p$.
$\boldsymbol{W}^{\mathsf{norm}} \leftarrow \boldsymbol{V}\boldsymbol{\Lambda}^{-1/2}$
$\boldsymbol{W}^{\mathsf{eff}} = \boldsymbol{W}^{\mathsf{reg}}\boldsymbol{W}^{\mathsf{norm}}$
$\boldsymbol{A}^{\mathsf{eff}} \leftarrow \boldsymbol{W}^{\mathsf{eff}T}\boldsymbol{A}$

[4] The optimization of [CK14] reduces the attack complexity to $\Theta\left(p(n_s+2^b)\right)$ at the cost of $\Theta\left(2^b p\right)$ memory, which (even with enough memory) is not efficient due to memory bandwidth saturation.

First, for the regression, we have to solve the linear system:

$$\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B},$$

for the unknown $\boldsymbol{A}$ with:

$$\boldsymbol{B} = \boldsymbol{\beta}(\boldsymbol{x})\boldsymbol{\beta}(\boldsymbol{x})^T = \sum_{i=1}^{n_p} \boldsymbol{\beta}(x_i)\boldsymbol{\beta}(x_i)^T,$$
$$\boldsymbol{C} = \boldsymbol{L}\boldsymbol{\beta}(\boldsymbol{x})^T = \sum_{i=1}^{n_p} \boldsymbol{l}_i\boldsymbol{\beta}(x_i)^T,$$

where $\boldsymbol{B}$ is a full-rank square matrix if the span of the set of bit-vectors x_i is $\mathbb{F}^b$.[5] This system is therefore a re-writing of the normal equations (Equation 4) in a matrix form.

Let us remark that $\boldsymbol{B}$ and $\boldsymbol{C}$ can be computed incrementally while the profiling traces are being acquired or loaded. For improved implementation efficiency, one may also decompose $\boldsymbol{x}$ and $\boldsymbol{L}$ in small blocks rather than single elements (resp., columns).

Next, we derive an efficient formula for the inter-class scatter matrix $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$, avoiding the sum over the 2^b classes, as formalized by the following proposition.

Proposition 1 (Efficient $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$ formula)**.**

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} = \boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^T - n_p\hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T.$$

Proof. First, we observe that

$$\sum_{x\in\mathcal{X}} |\mathcal{L}_x|\,\mathbf{m}(x) = \sum_{i=1}^{n_p} \mathbf{m}(x_i) = \sum_{i=1}^{n_p} \boldsymbol{A}\boldsymbol{\beta}(x_i) = \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})\mathbb{1}_{n_p},$$

is the first column of $\boldsymbol{A}\boldsymbol{B}$, since the first column of $\boldsymbol{\beta}(\boldsymbol{x})^T$ is $\mathbb{1}_{n_p}$ by construction (the first basis function is the constant 1). Therefore, it is equal to the first column of $\boldsymbol{C}$, which is itself equal to $\boldsymbol{L}\mathbb{1}_{n_p} = \sum_{i=1}^{n_p} \boldsymbol{l}_i$. Hence

$$\sum_{x\in\mathcal{X}} |\mathcal{L}_x|\,\mathbf{m}(x) = n_p\hat{\boldsymbol{\mu}}$$

by definition of $\hat{\boldsymbol{\mu}}$. We then expand the definition of $\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}$:

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} = \sum_{x\in\mathcal{X}} |\mathcal{L}_x| \left[\mathbf{m}(x)\mathbf{m}(x)^T - \mathbf{m}(x)\hat{\boldsymbol{\mu}}^T - \hat{\boldsymbol{\mu}}\mathbf{m}(x)^T + \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T\right],$$

which, using the previous result, gives

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}} = \sum_{i=1}^{n_p} \mathbf{m}(x_i)\mathbf{m}(x_i)^T - n_p\hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T.$$

We conclude the proof by remarking that

$$\sum_{i=1}^{n_p} \mathbf{m}(x_i)\mathbf{m}(x_i)^T = \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))^T.\,(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))\,(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))^T = \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})\boldsymbol{\beta}^T(\boldsymbol{x})\boldsymbol{A}^T = \boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^T.$$

□

[5] For example, this happens with overwhelming probability for the regression with the linear basis when x is selected uniformly at random and the profiling set is large enough.

And for $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$, we similarly derive the following incremental formula.

Proposition 2 (Efficient $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ formula)**.**

$$\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} = \boldsymbol{L}\boldsymbol{L}^T - \boldsymbol{C}\boldsymbol{A}^T - \boldsymbol{A}\boldsymbol{C}^T + \boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^T.$$

Proof. We expand the definition of $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$:

$$\begin{aligned}\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} &= \sum_{i=0}^{n_p} \left[\boldsymbol{l}_i\boldsymbol{l}_i^T - \boldsymbol{l}_i\mathbf{m}(x_i)^T - \mathbf{m}(x_i)\boldsymbol{l}_i^T + \mathbf{m}(x_i)\mathbf{m}(x_i)^T\right], \\ &= \boldsymbol{L}\boldsymbol{L}^T - \boldsymbol{L}(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))^T - \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})\boldsymbol{L}^T + \boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x})(\boldsymbol{A}\boldsymbol{\beta}(\boldsymbol{x}))^T,\end{aligned}$$

and the claim then follows from the definitions of $\boldsymbol{B}$ and $\boldsymbol{C}$. □

The matrix $\boldsymbol{L}\boldsymbol{L}^T$ can be computed incrementally as $\sum_{i=1}^{n_p} \boldsymbol{l}_i\boldsymbol{l}_i^T$. Exploiting such a proposition nevertheless requires some attention since it may not always be numerically stable if a lot of traces are needed for profiling. In practice though, assuming that the adversary/evaluator measures traces made of 8-bit or 16-bit integers and the accumulator is a 64-bit integer, the computation remains exact as long as $|\mathcal{L}| < 2^{32}$).[6]

Once the scatter matrices are known, one can partially solve the generalized eigenproblem:

$$\boldsymbol{S}_{\boldsymbol{B}}^{\mathsf{reg}}\boldsymbol{w}_i = \lambda_i \boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}\boldsymbol{w}_i, \tag{9}$$

to get the projection matrix $\boldsymbol{W}^{\mathsf{reg}} = (\boldsymbol{w}_1 \cdots \boldsymbol{w}_p)$. This is fairly efficient since both scatter matrices are symmetric semi-positive definite, and we usually only need a small part of the spectrum (the one corresponding to the $p \ll n_s$ largest eigenvalues). Moreover, if the traces are noisy and $n_p > n_s$, the matrix $\boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}}$ is non-singular with probability 1, which makes the problem easy to solve [PW69]. At this stage, we could compute $\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}}^{-1}$ and then use Equation 7. However, in order to further simplify the model, we will modify the projection to get a Gaussian distribution with unit covariance. Namely, we compute the eigendecomposition of the symmetric positive-definite (if we profile with at least p noisy traces) $p \times p$ matrix: $\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}} = |\mathcal{L}|^{-1}\, \boldsymbol{W}^{\mathsf{reg}T} \boldsymbol{S}_{\boldsymbol{W}}^{\mathsf{reg}} \boldsymbol{W}^{\mathsf{reg}}$:

$$\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues and $\boldsymbol{V}$ is the orthonormal matrix of eigenvectors. Then, we set $\boldsymbol{W}^{\mathsf{norm}} = \boldsymbol{V}\boldsymbol{\Lambda}^{-1/2}$. The final projection matrix is $\boldsymbol{W}^{\mathsf{eff}} = \boldsymbol{W}^{\mathsf{reg}}\boldsymbol{W}^{\mathsf{norm}}$, and we compute $\boldsymbol{A}^{\mathsf{eff}} = \boldsymbol{W}^{\mathsf{eff}T}\boldsymbol{A}$. As a result,

$$\boldsymbol{W}^{\mathsf{eff}T}\boldsymbol{l}^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x) = \boldsymbol{W}^{\mathsf{norm}T}\left(\boldsymbol{W}^{\mathsf{reg}T}(\boldsymbol{l}^* - \mathbf{m}(x))\right)$$

and since $\boldsymbol{W}^{\mathsf{norm}}\boldsymbol{W}^{\mathsf{norm}T} = \hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}}^{-1}$, we can re-write Equation 7 as

$$\hat{\mathsf{f}}^{\mathsf{RLDA}}[\boldsymbol{l}^*|X = x] = \alpha \exp\left(-\frac{1}{2}\left\|\boldsymbol{W}^{\mathsf{eff}T}\boldsymbol{l}^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2\right). \tag{10}$$

Let us note that we do not need to compute $\alpha = 1/\sqrt{(2\pi)^p|\hat{\Sigma}_{\boldsymbol{W}^{\mathsf{reg}}}|}$ since we only have to know $\hat{\mathsf{f}}^{\mathsf{RLDA}}[\boldsymbol{l}^*|X = x]$ up to a constant factor to apply Bayes rule.

Profiling complexity. Overall, RLDA profiling using Algorithm 1 has a computational complexity of $\Theta\left(n_s^2(b + n_p)\right)$ and memory usage $\Theta\left(n_s^2\right)$.[7] Remarkably, and contrary to the baseline method of Subsection 2.3, this complexity is not exponential in b.

[6] If numerical precision is a concern, an easy improvement is to ensure that the mean of the traces is close to zero, which can be done by computing the mean of a few traces and subtracting it to every trace.

[7] We simplified these expressions by assuming that $b \leq n_s$ and $b \leq n_p$.

Attack complexity. For one attack trace $\boldsymbol{l}^*$, computing the full distribution of X using Equation 10 and Equation 1 has a computational cost $\Theta\left((n_s + 2^b)p\right)$, instead of $\Theta\left(2^b(b+p)n_s\right)$ for a naive implementation, since we can amortize the cost of computing $\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)$ to $\Theta\left(p\right)$ on average by caching some intermediate sums. The main memory usage is dominated by the storage of the resulting distribution which has cost $\Theta\left(2^b\right)$, while the cache used for the above optimization is small: $\Theta\left(pb\right)$. When n_a traces are available for the attack, we compute Equation 10 independently for each trace and multiply the resulting distributions, which has computational complexity $\Theta\left(n_a(n_s + 2^b)p\right)$.

Multi-trace optimisation. There are cases where multiple traces correspond to the same value of the variable X. For example, there is only one value for X in the Simple Power Analysis (SPA) setting, and the number of attack traces n_a can be larger than the number of possible values for X in a traditional differential analysis (DPA) setting. In such cases, the following optimization reduces the computation cost of computing the RLDA on n_a traces that share a common X value to the cost of a single-trace attack, up to the cost of summing all the attack traces together, i.e., time complexity of $\Theta\left(n_a n_s + (n_s + 2^b)p\right)$.

It is well-known that Equation 10 can be simplified by removing the quadratic term in $\boldsymbol{l}^*$ from the sum [CK13], which is what makes LDA "linear":

$$\left\|\boldsymbol{W}^{\mathsf{eff}\,T}\boldsymbol{l}^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2 = \left\|\boldsymbol{W}^{\mathsf{eff}\,T}\boldsymbol{l}^*\right\|^2 - 2(\boldsymbol{W}^{\mathsf{eff}\,T}\boldsymbol{l}^*)^T\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x) + \left\|\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2.$$

This scales the terms by a factor independent of x, which disappears when applying the normalization of Equation 1, leading to a formula linear in $\boldsymbol{l}^*$. Then, the product of the distributions for the n_a traces is turned into a sum of the exponents. We revisit this technique by keeping the exponent in the form of a squared norm, which makes it very efficient to compute and avoids numerical overflows in the exponential (which otherwise would require a normalization pass before computing the exponential to avoid overflows), while maintaining the explicit computations of probabilities to discriminate the key (contrary to [CK13]). Our technique is derived as follows:

$$\begin{aligned}\hat{\mathsf{f}}^{\mathsf{RLDA}}[(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)|X=x] &= \prod_{i=1}^{n_a}\alpha\exp\left(-\frac{1}{2}\left\|\boldsymbol{W}^{\mathsf{eff}\,T}\boldsymbol{l}_i^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2\right),\\ &= \alpha_{(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)}\exp\left(-\frac{1}{2n_a}\left\|\boldsymbol{W}^{\mathsf{eff}\,T}\sum_{i=1}^{n_a}\boldsymbol{l}_i^* - n_a\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2\right), \quad (11)\end{aligned}$$

where the normalization factor $\alpha_{(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)}$ does not have to be computed, as it cancels out in Equation 1. The correctness of Equation 11 follows from the equality:

$$\sum_{i=1}^{n_a}\left\|\boldsymbol{W}^{\mathsf{eff}\,T}\boldsymbol{l}_i^* - \boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2 = \frac{1}{n_a}\left\|\boldsymbol{W}^{\mathsf{eff}\,T}\left(\sum_{i=1}^{n_a}\boldsymbol{l}_i^*\right) - n_a\boldsymbol{A}^{\mathsf{eff}}\boldsymbol{\beta}(x)\right\|^2 + \omega_{(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)},$$

where $\omega_{(\boldsymbol{l}_1^*,\ldots,\boldsymbol{l}_{n_a}^*)}$ depends only on $\boldsymbol{l}_i^*$ and $\boldsymbol{W}^{\mathsf{eff}}$.

3.2 Efficient PI bound

The previous section described a tool enabling to evaluate a model for large (e.g., 32-bit) target intermediate values. As an adversary/evaluator, such a tool can directly be used to mount improved attacks, which we will further detail in Section 5. But as an evaluator, it may also happen that one is just interested to gauge the security level of an implementation, without mounting explicit attacks. This is typically what happens when using shortcut formulas like [DFS19, dCGRP19] for divide-and-conquer attacks

and [GGSB20] for analytical attacks. In this case, the evaluation problem essentially reduces to the estimation of an information theoretic metric like the PI.

Looking at Equation 8, estimating the PI by sampling boils down to estimate a model $|\mathcal{L}'|$ times – the value of $|\mathcal{L}'|$ depending on the security level. However, in order to turn the leakage (conditional) likelihoods $\hat{\mathsf{f}}[\boldsymbol{l}|x]$ into probabilities $\hat{\mathsf{p}}[x|\boldsymbol{l}]$ for the target intermediate value, Bayes's rule must be applied. This implies evaluating the model over all the possible classes $x' \in \mathcal{X}$, where $\mathcal{X}$ grows exponentially with the number of bits b of X. For intermediate target sizes, like the 32-bit ones we consider in this work, it may result in a situation where despite feasible, evaluating the model exhaustively $\Theta\left(|\mathcal{L}'| \cdot |\mathcal{X}|\right)$ times becomes cumbersome. In this section, we therefore propose a way to improve the efficiency of the PI estimation, by replacing its exact computation with cheaper bounds. Precisely, we show a solution to reduces the $\Theta\left(|\mathcal{L}'| \cdot |\mathcal{X}|\right)$ computational complexity of the exact estimation down to $\Theta\left(|\mathcal{L}'| \cdot S + |\mathcal{X}| \log S\right)$ for the approximated one, at a $\Theta\left(S\right)$ memory cost, where the parameter S determines the tightness of the bounds.

Concretely, instead of computing $\hat{\mathsf{p}}[x|\boldsymbol{l}]$ exactly for every x, we will bound it by clustering some x values according to some metric. These bounds will then be translated into bounds on the estimated PI. More precisely, using the RLDA model, we have:

$$\hat{\mathsf{f}}[X = x|\boldsymbol{l}] = \frac{\exp(-\frac{1}{2}\left\|l - \mathsf{m}(x)\right\|^2)}{\sum_{x'=0}^{2^b-1} \exp(-\frac{1}{2}\left\|l - \mathsf{m}(x')\right\|^2)}.$$

Our core idea is that if $\mathsf{m}(x')$ and $\mathsf{m}(x'')$ are close to each other, then the corresponding terms in the denominator will have close values, hence we can compute the term $\exp(-\frac{1}{2}\left\|l - \mathsf{m}(x')\right\|^2)$ for x' and use it to infer bounds for the term $\exp(-\frac{1}{2}\left\|l - \mathsf{m}(x'')\right\|^2)$. As a result, our goal (to gain efficiency) is to find many x'' classes close to each class x' for which we compute the exponentiation. For this purpose, we perform a preprocessing to identify the classes that are close to each other, then compute all the relevant terms, and finally put these results together to get bounds for all $\hat{\mathsf{f}}[X = x|\boldsymbol{l}]$ values.

The preprocessing is parameterized by a distance threshold $t \geq 0$ and builds a set $\mathcal{X}' \subset \mathcal{X}$ and a map $\sigma : \mathcal{X} \to \mathcal{X}'$ such that $\left\|\mathsf{m}(x) - \mathsf{m}(\sigma(x))\right\| \leq t$ for every $x \in \mathcal{X}$. Each $x' \in \mathcal{X}'$ is named a cluster center, and the pre-images of x' form a cluster. For every cluster center, we compute the size of its cluster $s(x')$. The construction of $\mathcal{X}'$ and $s(\cdot)$ is done according to Algorithm 2, where the map σ is implicit: $\sigma(x)$ is the nearest point to x at the time x is processed. Its correctness can be verified by observing that the expected properties on $\sigma(.)$ and $s(.)$ are kept satisfied at every iteration (when considering only the subset of values $x \in \mathcal{X}$ that have already been processed). We illustrate the result of the clustering in Figure 2. Let $S = |\mathcal{X}'|$, the time complexity of this algorithm is $\mathcal{O}\left(|X| \cdot \log S\right)$ and its memory complexity is $\mathcal{O}\left(\mathcal{X}\right)$ when computing σ^{-1}, or $\mathcal{O}\left(S\right)$ otherwise.

Once clustering is performed, we can compute probability bounds knowing only the cluster centers and the cluster sizes using the following proposition.

Proposition 3 (Probability bounding with clustering)**.** *Let $\mathcal{X}'$ and $s(\cdot)$ be the outputs of Algorithm 2 for the inputs $\mathcal{X}$, m and t, and let us denote $\alpha(y) = \exp(-\frac{1}{2}y)$. Then, for any $\boldsymbol{l} \in \mathbb{R}^p$:*

$$\sum_{x' \in \mathcal{X}'} s(x') \cdot l_{x'} \leq \sum_{x \in \mathcal{X}} \alpha\left(\left\|\boldsymbol{l} - \mathsf{m}(x)\right\|^2\right) \leq \sum_{x' \in \mathcal{X}'} s(x') \cdot u_{x'},$$

where:

$$l_{x'} = \alpha\left(\left(\left\|\boldsymbol{l} - \mathsf{m}(x')\right\| + t\right)^2\right),$$
$$u_{x'} = \alpha\left(\left(\max\left(0, \left\|\boldsymbol{l} - \mathsf{m}(x')\right\| - t\right)\right)^2\right).$$

Algorithm 2 Clustering.

Input: A set of classes $\mathcal{X}$, a model $\mathsf{m} : \mathcal{X} \to \mathbb{R}^p$ and a threshold distance t.
Output: A set of cluster centers $\mathcal{X}'$ (represented by nn in the algorithm), cluster sizes $s(\cdot)$ and list of classes for each cluster, next denoted as σ^{-1}.

▷ `NN` is a set of elements in $(\mathcal{X}, \mathbb{R}^p)$ with logarithmic cost (in its size) `insert` and $\mathtt{nn} : \mathbb{R}^p \to \mathcal{X}$ (nearest neighbor) query (retrieves the stored $x \in \mathcal{X}$ that corresponds to closest stored vector). We instantiate it with a k-d tree [Ben75].

$\mathcal{X}' \leftarrow \emptyset$
$s \leftarrow$ `Map.newEmpty()`
$nn \leftarrow$ `NN.newEmpty()`
for $x \in \mathcal{X}$ **do**
  $x' \leftarrow nn.\mathtt{nearest}(\mathsf{m}(x))$
  if $\|\mathsf{m}(x) - \mathsf{m}(x')\| > t$ **then**
    $\mathcal{X}' \leftarrow \mathcal{X}' \cup \{x\}$
    $nn.\mathtt{insert}(x, \mathsf{m}(x))$
    $s(x) \leftarrow 0$
    $\sigma^{-1}(x) \leftarrow \emptyset$
    $x' \leftarrow x$
  $s(x') \leftarrow s(x') + 1$
  $\sigma^{-1}(x') \leftarrow \sigma^{-1}(x') \cup \{x\}$ ▷ Only when using Equation 13 and not Equation 12.

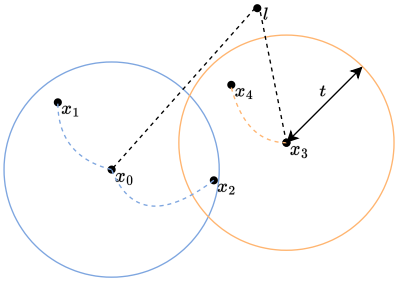

Figure 2: Illustration of the clustering-based PI computation for the set of classes $\mathcal{X} = \{x_0, \ldots, x_4\}$. The cluster centers are $\mathcal{X}' = \{x_0, x_3\}$ in our example, and the bound from Proposition 3 is $3l_{x_0} + 2l_{x_3} \leq \sum_{x \in \mathcal{X}} \alpha\left(\|\boldsymbol{l} - x_i\|^2\right) \leq 3u_{x_0} + 2u_{x_3}$.

Proof. Let $x \in \mathcal{X}$ and $x' = \sigma(x)$ (hence $\|\mathsf{m}(x) - \mathsf{m}(x')\| \leq t$), using the triangular inequality we get:

$$\|\boldsymbol{l} - \mathsf{m}(x)\|^2 \leq \left(\|\boldsymbol{l} - \mathsf{m}(x')\| + \|\mathsf{m}(x) - \mathsf{m}(x')\|\right)^2 \leq \left(\|\boldsymbol{l} - \mathsf{m}(x')\| + t\right)^2.$$

We proceed similarly to calculate the lower bound. Starting from the triangular inequality $\|\boldsymbol{l} - \mathsf{m}(x')\| \leq \|\boldsymbol{l} - \mathsf{m}(x)\| + \|\mathsf{m}(x) - \mathsf{m}(x')\|$, we get $\|\boldsymbol{l} - \mathsf{m}(x)\| \leq \|\boldsymbol{l} - \mathsf{m}(x')\| - t$, and then $\|\boldsymbol{l} - \mathsf{m}(x)\|^2 \leq \left(\max\left(0, \|\boldsymbol{l} - \mathsf{m}(x')\| - t\right)\right)^2$.

We now have $\alpha^{-1}\left(u_{x'}\right) \leq \|\boldsymbol{l} - \mathsf{m}(x)\|^2 \leq \alpha^{-1}\left(l_{x'}\right)$, which leads to the expected result by observing that $\alpha(\cdot)$ is a decreasing function and then summing over all $x \in \mathcal{X}$. □

Corollary 1. *With the same hypotheses as Proposition 3, and assuming that:*

$$\hat{\mathsf{f}}[X = x|\boldsymbol{l}] = \frac{\alpha\left(\|l - \mathsf{m}(x)\|^2\right)}{\sum_{x' \in \mathcal{X}} \alpha\left(\|l - \mathsf{m}(x')\|^2\right)},$$

the following holds:

$$\frac{\alpha\left(\|l-\mathsf{m}(x)\|^2\right)}{\sum_{x'\in\mathcal{X}'} s(x')\cdot u_{x'}} \leq \hat{\mathsf{f}}[X=x|\boldsymbol{l}] \leq \frac{\alpha\left(\|l-\mathsf{m}(x)\|^2\right)}{\sum_{x'\in\mathcal{X}'} s(x')\cdot l_{x'}}\,. \tag{12}$$

Proof. The two inequalities are trivial consequences of Proposition 3. □

We can then bound the PI by using the probability bounds of Corollary 1 in Equation 8, since this formula is increasing with $\hat{\mathsf{p}}[X=x|\boldsymbol{l}]$. The trade-off between the tightness of the bound and the computation cost is controlled with the parameter t.

When this trade-off is not satisfactory, we next propose an improvement of the method. It is based on the observation that the largest contributors to the gap between the probability upper- and lower-bounds are the clusters that are near to $\boldsymbol{l}$. Indeed, the value of $l_{x'}$ and $u_{x'}$ are small for cluster centers x' that are far from $\boldsymbol{l}$, compared to the closer ones, due to exponential shrining with $\|\boldsymbol{l}-\mathsf{m}(x')\|^2$. Therefore, even a large relative gap $u_{x'}/l_{x'}$ for these far clusters does not translate into a large difference on the overall sums $\sum_{x'\in\mathcal{X}'} s(x')\cdot l_{x'}$ and $\sum_{x'\in\mathcal{X}'} s(x')\cdot u_{x'}$. Conversely, the gap on the closest clusters will have the most impact on the overall tightness of the final bounds. We therefore propose to nullify this gap by computing the exact probability $\hat{\mathsf{f}}[\boldsymbol{l}|X=y]$ for all classes y in the close clusters, instead of relying on the bounds. Concretely, we implement this by storing list of all classes in each cluster when running Algorithm 2 (i.e., we store the map $\sigma^{-1}:\mathcal{X}'\to\mathcal{P}(\mathcal{X})$).[8] Then, when evaluating probability bounds for some leakage $\boldsymbol{l}$, we look in the *nn* structure (from Algorithm 2) for the cluster centers that are the closest to $\boldsymbol{l}$, and add them to a set $\mathcal{X}''$, stopping before the total size of the associated clusters exceeds a computational cost threshold ζ (i.e., we ensure $\sum_{x''\in\mathcal{X}''} s(x'')\leq\zeta$).[9] Finally, we use the following bounds:

$$\frac{\alpha\left(\|l-\mathsf{m}(x)\|^2\right)}{Z+\sum_{x'\in\mathcal{X}'\setminus\mathcal{X}''} s(x')\cdot u_{x'}} \leq \hat{\mathsf{f}}[X=x|\boldsymbol{l}] \leq \frac{\alpha\left(\|l-\mathsf{m}(x)\|^2\right)}{Z+\sum_{x'\in\mathcal{X}'\setminus\mathcal{X}''} s(x')\cdot l_{x'}}\,, \tag{13}$$

where:

$$Z=\sum_{y\in\mathcal{Y}}\alpha\left(\|l-\mathsf{m}(y)\|^2\right), \quad \text{with} \quad \mathcal{Y}=\bigcup_{x''\in\mathcal{X}''}\sigma^{-1}(x'').$$

Remark. We assumed a uniformly distributed X. The method can be adapted to a non-uniform X by storing the total probability of the clusters in $s(\cdot)$ instead of their sizes.

4 Atomic experiments

In this section, we investigate the ability of RLDA to exploit the leakage of a 32-bit ALU for a single isolated instruction and we analyze the obtained PI bounds. We compare the PI of the RLDA model with the one of state-of-the art models: LDA, LR and FT.

4.1 Experimental setup

We designed this first experiment to produce well isolated and easy to interpret leakages. The target 32-bit ARM instruction is a single XOR between two values stored in registers. As shown in Figure 3, this instruction is surrounded by `nop` instructions to separate the leakage of this instruction from other leakages (e.g., caused by memory accesses). The

[8] Which significantly increases the memory usage of the algorithm but not its computational complexity.
[9] Taking ζ as a small multiple of the number of clusters $|\mathcal{X}'|$ works well in practice.

XOR operands are independent uniform random values, and the target value is the result of the operation. As a result, the leakage of the operands cannot be exploited by the RLDA (like in a 1st-order masked implementation). The register writeback leakage is also reduced by overwriting an operand: this produces mainly a transition leakage which is then equivalent to leakage from the other operand. The impact of a more realistic setting including pipeline noise will be discussed with an improved SASCA in the next section.

```
ldrd r5,r6,[r0] // Load 2 random b-bit values (other bits set to 0)
NOP6 // Macro that generates 6 nop instructions, to flush the pipeline.
bl <trigger_up>
NOP6
eor.w r5,r5,r6 // Target instruction
NOP6
bl <trigger_low>
```

Figure 3: Target ARM assembly code for the atomic experiment.

Measurements were performed on the ChipWhisperer CW308 board, with an STM32F303 target (Cortex-M4, 3-stage pipeline). The target was running an 80 MHz clock derived internally from a 8 MHz external crystal. We used the Tektronix CT-1 AC current probe and we sampled our signal with the Picoscope 6424E at 625 MSample/s with 12-bit resolution. For each experiment, we used 1.3 million profiling traces and 1000 evaluation traces.

4.2 Analyzing various bus sizes

We analyzed the PI of the RLDA model (with a linear basis) when profiling the result of the XOR operation for different simulated bus sizes by keeping some bits of the operands set at 0. We profiled with multiple subspace dimensionalities: $p = 1, 2, 3, 4$ using the full leakage trace $n_s = 500$ samples (i.e., no POI selection). We also evaluated the tightness of our PI bound by running it with multiple clustering thresholds $t = 0.05, 0.15, 0.25, 0.35$, and aborting if the number of clusters $|\mathcal{X}'|$ exceeds 2^{20}, to limit the computational cost. We allowed the exact calculation of the likelihood for $\zeta = \max(2^{16}, 16 \cdot |\mathcal{X}'|)$ classes, which leads to a reasonnable overhead in execution time when there are many clusters and ensures that we calculate the exact PI when the cost to do so is low. The results are in Figure 4, where we add the MI of a Hamming Weight (HW) model for comparison purposes.

We first observe that the PI of the model grows with the number of target bits b (the entropy of the target variables increases as well). For $p = 1$, the PI of our model is close the to the MI of HW leakages, and it exceeds it for all values of b when $p \geq 2$. Moreover, the PI of the model increases with p (for any fixed $b > p$), but saturates quickly: the improvement between $p = 3$ and $p = 4$, which matches the intuition that most the leakage lies in a low-dimentional subspace.[10] Next, regarding the tightness of the clustering-based PI bounds, we observe that it decreases as the threshold t increases. This leads to an accuracy vs. efficiency trade-off, since the efficiency mostly depends on the number of clusters (reported in Table 1). This number is limited by two parameters: the total number of classes 2^b, and the number of clusters needed to cover the space of possible leakage values. Intuitively, this space is p-dimensional and the extreme values depend on the SNR of the leakage in this projected space (the noise variance is normalized to 1), while we try to cover it with balls of radius t. As for the computation cost: for the largest case ($b = 32$, $p = 4$), the RLDA profiling runs in about 18 s, the clustering takes approximately 4 h and the PI bounds computation using the evaluation traces takes about 10 s. The

[10] We did not study larger p values due to limitations of our PI estimation method (the clustering with k-d tree becomes less efficent). However, the RLDA is not limited to such low dimensions: its main limit is the increased risk of over-fitting, which can be limited with a sufficiently large profiling dataset.

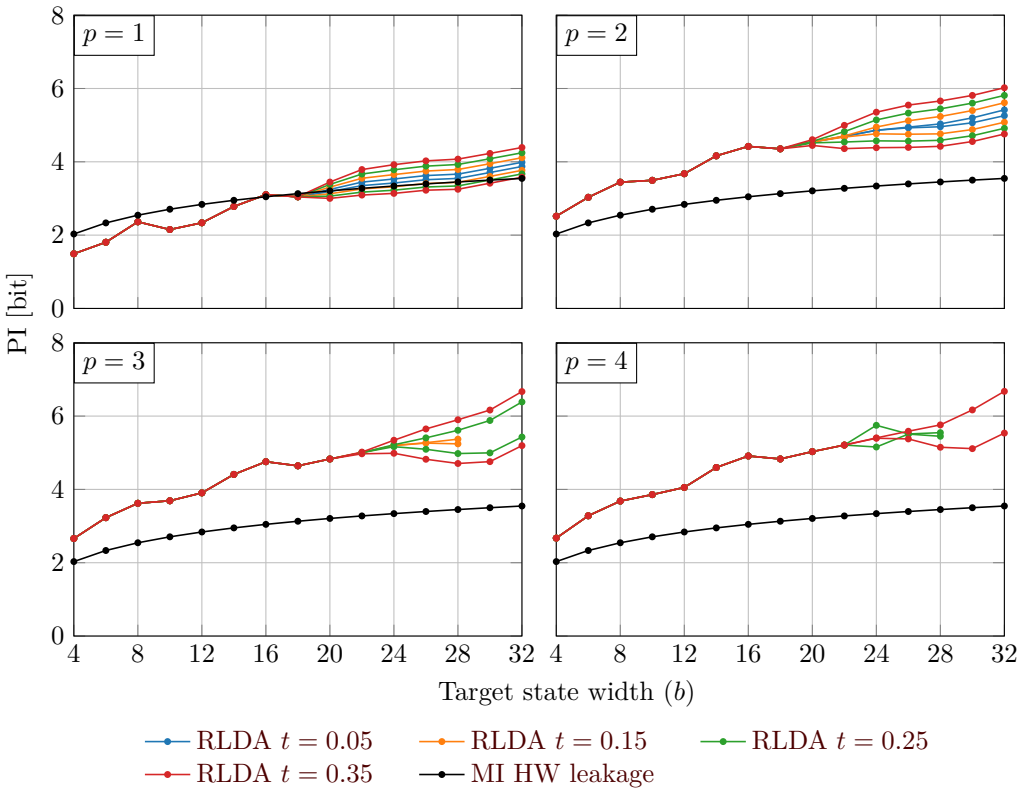

Figure 4: PI bounds for a XOR instruction leakage profiled with the RLDA model, for various values of the model parameters b, p, and of the PI estimation clustering parameter t. Identically colored lines represent the upper- and the lower-bound on the PI. The black line corresponds to the MI obtained from simulated Hamming weight leakages.

Table 1: Logarithm of the number of clusters ($\log_2(\mathcal{X}')$) for the atomic experiment. The cell color represents the ratio $|\mathcal{X}'| / |\mathcal{X}|$: darker cells reflect less effective clustering.

b	p	t				p	t			
		0.05	0.15	0.25	0.35		0.05	0.15	0.25	0.35
8	1	6.9	6.0	5.3	5.0	2	8.0	7.6	7.4	7.3
16		9.5	8.1	7.4	6.9		14.8	12.8	11.5	10.9
24		10.5	8.8	8.1	7.8		17.5	14.7	13.3	12.4
32		11.2	9.6	8.8	8.4		19.0	16.0	14.6	13.6
8	3	8.0	7.6	7.6	7.6	4	8.0	7.6	7.6	7.6
16		15.9	14.6	13.2	12.5		16.0	15.1	14.6	13.5
24			18.0	16.1	14.9				19.0	16.8
32				18.3	16.9					19.6

implementation is multi-threaded, with the exception of the clustering and was run on a 2.7 GHz AMD CPU. The k-d tree implementation has not been optimized.

4.3 Comparison with related leakage models

We also compared the RLDA with other profiled leakage models. Namely, we considered the LDA of Subsection 2.2 profiled using all the 500 samples of the traces (like RLDA), the LR-based profiling of Subsection 2.1 combined with a POI selection taking the p points with the highest SNR in the traces (POI selection is needed in this case, since we were not able to accurately estimate the large covariance matrices without this preprocessing), and the FT of Subsection 2.4 with 8-bit fragments (the PI reported is then the sum of the PI on all fragments) — this model is denoted as FT8. For completeness, we also analyzed FT with 1-bit fragments (denoted as FT1) and FT with 8-bit fragments marginalized to single)-bit variables (denoted as mFT8). The results for the cases $b = 16$ and $b = 32$ are shown in Figure 5. For the 16-bit bus, all models are analyzed. For the 32-bit case, we could only run FT and RLDA, since the other have a too high computational cost.

Starting with the 16-bit bus experiment, the PI of the RLDA can be evaluated exactly (i.e., no clustering is needed) and the resulting model achieves the highest PI of all models. The gains over FT are expected since these models artificially increase the amount of algorithmic noise due to their fragmentation process. The gains over the LR-based attack are presumably due to the suboptimal selection of POIs (i.e., selecting multiple univariate POIs with the SNR does not ensure that their multivariate selection is good, for example due to correlated leakage samples that may bring limited additional information). As for the LDA, it performs worse than the RLDA due to higher class mean estimation errors. Despite the large total profiling dataset, each class mean estimation for the LDA relies on only 20 traces per class on average, since there are 2^{16} classes in total. For a similar reason, the PI of the LDA model slowly decreases after $p = 6$. This effect does not appear for the RLDA, thanks to the regularization effect of the linear regression.

When moving to the 32-bit state, we need the clustering to estimate the PI of the RLDA, leading to uncertainty on the value of the PI for the RLDA. As in the 16-bit case, we observe that FT performs quite poorly compared to RLDA. The gap between the bounds for RLDA becomes wide for $p > 2$, and we could not compute meaningful bounds for $p > 6$. However, the results with the 16-bit bus suggest that there is not much information to gain by increasing p significantly beyond 5. Besides, and despite we cannot cheaply estimate the PI in those cases, we can still use the resulting models to run attacks.

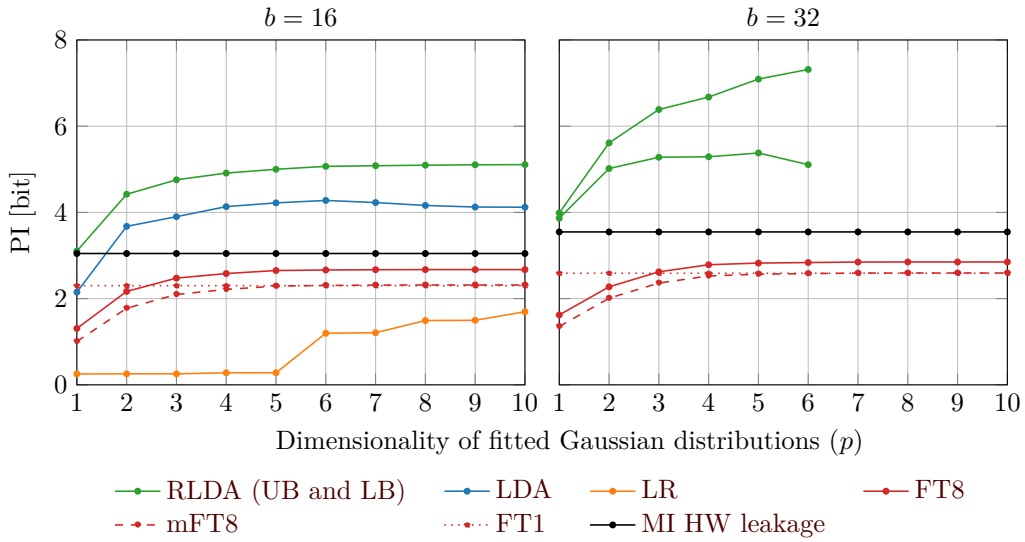

Figure 5: Comparison of profiled models for 16- and 32-bit bus.

5 SASCA against 32-bit bitslice implementations

We now use the RLDA model in order to mount an attack against a 32-bit software implementation of ISAP-A [DEM+20]. More precisely, we focus on the initialization of ISAP's re-keying which constitutes an interesting target since it heavily relies on the fact that the implementation of the Ascon-p permutation is secure against Simple Power Analysis (SPA). Besides, the Ascon-p permutation allows for a fully in-register implementation, which is in contrast with the Keccak permutation considered in [YK21], that requires many (leaky) memory accesses, and the ISAP-K implementation considered in [BBC+20] that was only 16-bit (and therefore more leaky as well).

As a result, our goal in this section is to demonstrate a single-trace attack. In this context, it is crucial to extract as much information as possible from 32-bit leakages, which is enabled by RLDA. Concretely, we target the first round of the initial permutation of ISAP, whose input is the long-term key and a fixed IV. We profile all the states of the S-box layer using RLDA, then mount a SASCA with the factor graph of Figure 1, and we finally compute the rank of the correct key using the algorithm of [PSG16].

For this purpose, we first introduce a new efficient algorithm for the computation of the belief propagation on bitwise AND function nodes. While such $\Theta\left(b2^b\right)$ algorithms using fast transforms are well-known for modular additions and bitwise XOR [PPM17], we introduce a $\Theta\left(b2^b\right)$ algorithm for the AND belief propagation, which (to the best of our knowledge) is the first to improve over the state-of-the-art naive algorithm with complexity $\Theta\left(2^{2b}\right)$. This allows us to perform a SASCA with $b > 16$ in reasonable time.

In the rest of the section, we first give more details on the target implementation and our attack methodology. We then present and discuss the attack results.

5.1 Efficient belief propagation for AND gates

Let X, Y and Z be random variables in $\{0, \ldots, 2^b - 1\}$ such that $X \& Y = Z$, where & denotes the bitwise AND. Let $\boldsymbol{x} = (x_0, \ldots, x_{2^b-1}) \in \mathbb{R}^{2^b}$ denote the distribution of X ($\Pr[X = i] = x_i$), and similarly for Y and Z. To run the belief propagation, we have to

compute $f(\boldsymbol{x}, \boldsymbol{y})$, $g(\boldsymbol{z}, \boldsymbol{x})$ and $g(\boldsymbol{z}, \boldsymbol{y})$, where:

$$\begin{aligned} \boldsymbol{a} = f(\boldsymbol{x}, \boldsymbol{y}) \qquad & \text{iff} \qquad a_k = \sum_{i,j \text{ s.t. } i\&j = k} x_i y_j, \\ \boldsymbol{a} = g(\boldsymbol{z}, \boldsymbol{x}) \qquad & \text{iff} \qquad a_j = \sum_{i,k \text{ s.t. } i\&j = k} z_k x_i. \end{aligned}$$

We use the triangular matrices $U_0 = V_0 = \begin{pmatrix} 1 \end{pmatrix}$ and:

$$U_b = \begin{pmatrix} U_{b-1} & U_{b-1} \\ 0 & U_{b-1} \end{pmatrix}, \qquad V_b = \begin{pmatrix} V_{b-1} & 0 \\ V_{b-1} & -V_{b-1} \end{pmatrix},$$

whose inverses are $U_0^{-1} = V_0^{-1} = \begin{pmatrix} 1 \end{pmatrix}$:

$$U_b^{-1} = \begin{pmatrix} U_{b-1}^{-1} & -U_{b-1}^{-1} \\ 0 & U_{b-1}^{-1} \end{pmatrix}, \qquad V_b^{-1} = V_b.$$

For any vector $\boldsymbol{a} \in \mathbb{R}^{2^b}$, we use the short-hands $\boldsymbol{a}^U = U_b \boldsymbol{a}$ and $\boldsymbol{a}^V = V_b \boldsymbol{a}$.

Proposition 4 (SASCA AND result distribution)**.** *For any $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^{2^b}$,*

$$\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{y}) \quad \Leftrightarrow \quad \boldsymbol{z}^U = \boldsymbol{x}^U \odot \boldsymbol{y}^U,$$

where $\odot$ denotes the element-wise product.

Proof. Let us first remark that since U_b is invertible, both directions of the equivalence imply each other, therefore we prove the left-to-right implication.

We work by induction: we first verify the base case $b = 1$ by remarking that $f(\boldsymbol{x}, \boldsymbol{y}) = (x_0 y_0 + x_0 y_1 + x_1 y_0, x_1 y_1)$, which implies:

$$\boldsymbol{z}^U = (x_0 y_0 + x_0 y_1 + x_1 y_0 + x_1 y_1, x_1 y_1) = (x_0 + x_1, x_1) \odot (y_0 + y_1, y_1) = \boldsymbol{x}^U \odot \boldsymbol{y}^U.$$

Then, for any $b > 1$, we denote $\boldsymbol{x}|_0 = (x_0, \ldots, x_{2^{b-1}-1})$, $\boldsymbol{x}|_1 = (x_{2^{b-1}}, \ldots, x_{2^b-1})$, and similarly for $\boldsymbol{y}$ and $\boldsymbol{z}$. We remark that $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{y})$ implies:

$$\begin{aligned} \boldsymbol{z}|_0 &= f(\boldsymbol{x}|_0, \boldsymbol{y}|_0) + f(\boldsymbol{x}|_0, \boldsymbol{y}|_1) + f(\boldsymbol{x}|_1, \boldsymbol{y}|_0), \\ \boldsymbol{z}|_1 &= f(\boldsymbol{x}|_1, \boldsymbol{y}|_1), \end{aligned}$$

which, using the induction hypothesis and the linearity of the multiplication with U, gives:

$$\begin{aligned} \boldsymbol{z}|_0^U &= \boldsymbol{x}|_0^U \odot \boldsymbol{y}|_0^U + \boldsymbol{x}|_0^U \odot \boldsymbol{y}|_1^U + \boldsymbol{x}|_1^U \odot \boldsymbol{y}|_0^U, \\ \boldsymbol{z}|_1^U &= \boldsymbol{x}|_1^U \odot \boldsymbol{y}|_1^U. \end{aligned}$$

This can be re-written as:

$$\begin{aligned} \boldsymbol{z}|_0^U + \boldsymbol{z}|_1^U &= \boldsymbol{x}|_0^U \odot \boldsymbol{y}|_0^U + \boldsymbol{x}|_0^U \odot \boldsymbol{y}|_1^U + \boldsymbol{x}|_1^U \odot \boldsymbol{y}|_0^U + \boldsymbol{x}|_1^U \odot \boldsymbol{y}|_1^U, \\ \boldsymbol{z}|_1^U &= \boldsymbol{x}|_1^U \odot \boldsymbol{y}|_1^U. \end{aligned}$$

The first equation can be simplified to $\boldsymbol{z}|_0^U + \boldsymbol{z}|_1^U = (\boldsymbol{x}|_0^U + \boldsymbol{x}|_1^U) \odot (\boldsymbol{y}|_0^U + \boldsymbol{y}|_1^U)$, leading to:

$$\boldsymbol{z}^U = \begin{pmatrix} \boldsymbol{z}|_0^U + \boldsymbol{z}|_1^U \\ \boldsymbol{z}|_1^U \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}|_0^U + \boldsymbol{x}|_1^U \\ \boldsymbol{x}|_1^U \end{pmatrix} \odot \begin{pmatrix} \boldsymbol{y}|_0^U + \boldsymbol{y}|_1^U \\ \boldsymbol{y}|_1^U \end{pmatrix} = \boldsymbol{x}^U \odot \boldsymbol{y}^U.$$

□

Proposition 5 (SASCA AND operand distribution)**.** *For any* $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^{2^b}$,

$$\boldsymbol{y} = g(\boldsymbol{z}, \boldsymbol{x}) \quad \Leftrightarrow \quad \boldsymbol{y}^V = \boldsymbol{x}^U \odot \boldsymbol{z}^V.$$

Proof. The proof is very similar to the one of Proposition 4. We only explain the main difference in the induction argument (the base case is changes similarly). $\boldsymbol{y} = g(\boldsymbol{z}, \boldsymbol{x})$ implies:

$$\begin{aligned}\boldsymbol{y}|_0 &= g(\boldsymbol{z}|_0, \boldsymbol{x}|_0) + g(\boldsymbol{z}|_0, \boldsymbol{x}|_1),\\ \boldsymbol{y}|_1 &= g(\boldsymbol{z}|_0, \boldsymbol{x}|_0) + g(\boldsymbol{z}|_1, \boldsymbol{x}|_1),\end{aligned}$$

which is then re-written as:

$$\begin{aligned}\boldsymbol{y}^V|_0 = \boldsymbol{z}^V|_0 \odot \boldsymbol{x}|_0^U + \boldsymbol{z}^V|_0 \odot \boldsymbol{x}|_1^U = \boldsymbol{z}^V|_0 \odot (\boldsymbol{x}|_0^U + \boldsymbol{x}|_1^U),\\ \boldsymbol{y}^V|_0 - \boldsymbol{y}|_1^V = \boldsymbol{z}^V|_0 \odot \boldsymbol{x}|_1^U - \boldsymbol{z}|_1^V \odot \boldsymbol{x}|_1^U = (\boldsymbol{z}^V|_0 - \boldsymbol{z}|_1^V) \odot \boldsymbol{x}|_1^U\end{aligned}$$

leading to $\boldsymbol{y}^V = \boldsymbol{z}^V \odot \boldsymbol{x}^U$. □

Eventually, the computations $\boldsymbol{x}^U = U_b\boldsymbol{x}$ and $\boldsymbol{x}^V = V_b\boldsymbol{x}$ (as well as their inverses) can be performed in-place and in time $\Theta\left(b2^b\right)$ using an algorithm based radix-2 butterflies.

5.2 Attack target and methodology

Our attack target is the C reference implementation of the Ascon-p permutation running on a Cortex-M4 microcontroller compiled with optimizations.[11] We keep only the S-box computation in the trace and use the same side-channel acquisition setup as in Subsection 4.1. Each acquisition is repeated 100 times to give averaged traces that contain less noise, which is allowed by the threat model of ISAP. The initial state of the permutation is the initial state for the IsapRk (f = enc), that is: $(K_0, K_1, \mathrm{IV}_{\mathrm{KE}}, 0, 0)$ with $\mathrm{IV}_{\mathrm{KE}}$ = `0x01804001_0C01060C`. The permutation state is stored in 10 32-bit registers, which fits into the register file of the target. Hence, there is no register spilling, which would cause increased leakage. Each 64-bit word of state is split into two registers: the 32 LSBs and the 32 MSBs, that we refer to as the lower and upper parts of the states.

We used 100k profiling traces to build a RLDA model with $p = 10$ for both parts of the non-zero variables in the Ascon S-box, plus the distance between the key words (i.e., $K_0 \oplus K_1$). After getting the distribution for all the target variables using RLDA, we run a SASCA using the graph of Figure 1 twice, setting the IV to the known constant: one for the lower part and one for the upper part of the state. Each SASCA is executed with two belief propagation iterations, which is sufficient to propagate all the variables information to the key and limits the instability of the loopy belief propagation algorithm.

We compare our RLDA-based attack with the state of the art by running the same attack with the mFT8 model (marginalized 8-bit fragment templates) of [YK21], using the same parameters and set of attack and profiling traces as for the RLDA.

5.3 Attack results and discussion

We ran the attack on 6 randomly chosen keys, leading to the final key rank distributions shown in Table 2. For the RLDA model, after the SASCA, the correct key is within enumeration power (i.e., rank less than 2^{64}) in a majority of the cases. The rank of the key pre-SASCA (i.e., using only the leakage on the key words) is most of the time below 2^{70}, which remains within enumeration range for determined adversaries. We also observe a strong improvement of the ranks compared to fragment templates.

[11] https://github.com/ascon/ascon-c/blob/33abe4bee86/crypto_aead/ascon128av12/ref/round.h. We use `arm-none-eabi-gcc` version 9.2.1 with flags `-Os -mthumb -mcpu=cortex-m4`.

Table 2: Attack of a 32-bit ISAP-A implementation for 6 randomly chosen keys. The number given is the base 2 logarithm of the rank of the correct key.

	k_0	k_1	k_2	k_3	k_4	k_5
RLDA Pre-SASCA	63.9	69.4	70.6	67.4	69.4	65.2
RLDA Post-SASCA	51.3	62.0	61.1	66.3	74.3	48.2
mFT8 Pre-SASCA	89.5	92.7	101.8	92.7	104.6	77.7
mFT8 Post-SASCA	80.1	87.7	104.9	91.6	102.5	75.8

Table 3: PI bounds for profiled variables in the ISAP-A implem. ($p = 4$, $t = 0.5$).

Variable	$\mathrm{PI}_{\text{lower part}}$	$\mathrm{PI}^{\text{mFT8}}_{\text{lower part}}$	$\mathrm{PI}_{\text{upper part}}$	$\mathrm{PI}^{\text{mFT8}}_{\text{upper part}}$
K_0	9.918 ± 0.653	6.233	10.132 ± 0.575	5.204
K_1	15.162 ± 0.000	10.408	11.416 ± 0.175	6.897
A	14.853 ± 0.088	6.53	12.879 ± 0.297	4.91
C	8.726 ± 1.111	6.231	9.547 ± 0.805	5.13
L_1	8.485 ± 1.217	5.816	6.806 ± 1.830	5.046
L_2	7.904 ± 0.148	5.905	7.659 ± 0.143	4.706
L_3	8.087 ± 1.376	5.929	7.327 ± 1.640	5.12

We additionally computed the PI of our RLDA models (under the same settings as Subsection 4.2 but reduced to $p = 4$, hence they are worse than the models we used for the attacks), which are shown in Table 3. We observe that the PI is sometimes lower and sometimes higher than the one of the experiment in Subsection 4.2. Two opposing effects can explain these differences. First, the pipeline is now full, leading to multiple states leaking simultaneously and therefore generating more computational/algorithmic noise. Second, the increased amount of computation on every state increases the leakage. In particular, computations such as K_1&IV create leakage on a subset of the bits of a state, adding information on K_1. We note that the mFT8 has lower PI than the RLDA (by roughly 1.5 to 8 bits, depending on the variable), which explains the worse key ranks.

We finally discuss the execution time of the attacks, when run on 8 cores of a AMD Epyc 7501 2.0 GHz CPU. First, the profiling of the RLDA takes less than 2 minutes for all 14 target variables. Then, the computation of the distributions from the RLDA for one attack trace takes less than 15 second per variable. The belief propagation for one attack is performed in about 2.7 h (it is single-threaded). As for the memory usage, the most expensive step is the SASCA, for which our (far from optimal) prototype implementation stored 35 distributions of 32 GiB each, summing to 1.13 TiB of RAM.

6 Conclusions

In this paper we proposed optimized solutions to compute the RLDA profiled model, allowing it to scale up to 32-bit targets. We then used this model as the basis for a single-trace SASCA against an implementation of the Ascon-p S-box on a 32-bit microcontroller. An important application for such an attack is the ISAP-A re-keying. Our results indeed show for the first time that the SPA security on which this re-keying relies can be attacked when implemented on unprotected 32-bit microcontrollers. Our current results target the initialization of the re-keying (which could be pre-computed) and open various directions for further resarch. First, the RLDA model is fairly simple and it could be improved towards better generality and accuracy, which could be useful to attack less leaky targets than the Cortex-M4. Among possible directions, one could consider the use of a larger (e.g., quadratic) basis for the regression or the use of Quadratic Discriminant Analysis (QDA)

instead of LDA. Whether efficiently profiling long traces for large target intermediate values is possible with more advanced (e.g., deep learning) models is an interesting open problem as well. Besides, since our attack is bottelnecked by the SASCA, due to its heuristic nature and its time complexity, it would be interesting to investigate sparse and computationally efficient representations for distributions. This could help in growing the attacked states towards 64 bits, where computing full distributions is infeasible, and to extend our attacks beyond the initialization of ISAP's re-keying.

Acknowledgments

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Belgian Fund for Scientific Research (FNRS-F.R.S.) under convention 2.5020.11 and by the Walloon Region. Henri Devillez is a research fellow of the Belgian FRIA. François-Xavier Standaert is a senior research associate of the FNRS-F.R.S. This work and its presentation have been funded in parts by the ERC consolidator grant 724725 (acronym SWORD) and the ERC Advanced Grant 101096871 (acronym BRIDGE).

References

[BBB+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, 2020(S1):295–349, 2020.

[BBC+20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *CRYPTO (1)*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[BGP+20] Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Tedt, a leakage-resist AEAD mode for high physical security applications. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):256–320, 2020.

[BHM+19] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 713–737. Springer, 2019.

[BMPS21] Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. Improved leakage-resistant authenticated encryption based on hardware AES coprocessors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):641–676, 2021.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In *CARDIS*, volume 8419 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2013.

[CK14] Marios O. Choudary and Markus G. Kuhn. Efficient stochastic methods: Profiled attacks beyond 8 bits. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 85–103. Springer, 2014.

[CLM20] Valence Cristiani, Maxime Lecomte, and Philippe Maurine. Leakage assessment through neural estimation of the mutual information. In *ACNS Workshops*, volume 12418 of *Lecture Notes in Computer Science*, pages 144–162. Springer, 2020.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[dCGRP19] Eloi de Chérisey, Sylvain Guilley, Olivier Rioul, and Pablo Piantanida. Best information is most successful mutual information and success rate in side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):49–79, 2019.

[DEM+20] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. Isap v2.0. *IACR Trans. Symmetric Cryptol.*, 2020(S1):390–416, 2020.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[DFS19] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. *J. Cryptol.*, 32(4):1263–1297, 2019.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification, 2nd Edition*. Wiley, 2001.

[GGSB20] Qian Guo, Vincent Grosso, François-Xavier Standaert, and Olivier Bronchain. Modeling soft analytical side-channel attacks from a coding theory viewpoint. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):209–238, 2020.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In *COSADE*, volume 7275 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2012.

[KDB+22] Satyam Kumar, Vishnu Asutosh Dasu, Anubhab Baksi, Santanu Sarkar, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Side channel attack on stream ciphers: A three-step approach to state/key recovery. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):166–191, 2022.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.

[Man04] Stefan Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

[MCHS22] Loïc Masure, Gaëtan Cassiers, Julien M. Hendrickx, and François-Xavier Standaert. Information bounds and convergence rates for side-channel security evaluators. *IACR Cryptol. ePrint Arch.*, page 490, 2022.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *SPACE*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2016.

[PW69] G. Peters and James Hardy Wilkinson. Eigenvalues of ax = λbx with band symmetric A and B. *Comput. J.*, 12(4):398–404, 1969.

[RSV+11] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2011.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[USS+20] Florian Unterstein, Marc Schink, Thomas Schamberger, Lars Tebelmann, Manuel Ilg, and Johann Heyszl. Retrofitting leakage resilient authenticated encryption to microcontrollers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):365–388, 2020.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.

[YK21] Shih-Chun You and Markus G. Kuhn. Single-trace fragment template attack on a 32-bit implementation of keccak. In *CARDIS*, volume 13173 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021.